Python 代码整理汇总-1

Content

python Foundation

python之禅

```python
import this
```

```
The Zen of Python, by Tim Peters

```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

基础运算

```python
x = 5
print(type(x)) # 数据类型
print(x +2)
print(x - 2)
print(x * 2)
print(x / 2)
print(x ** 2)
print(x % 2)
```

```
<class 'int'>
7
3
10
2.5
25
1
```

```python
# Type Conversion, if unknown, help(str)
print(type(5))
print(type(5.0))
print(str(x)) # 字符串
print(int(3.5)) # 整数：仅保留整数部分
print(float(4)) # 浮点数
print(bool(3)) # 布尔值：非0都为True
```

```
<class 'int'>
<class 'float'>
5
3
4.0
True
```

数据结构

String

- 's' == "s" ; # for line comment, while '''s''' for multi-line comments 单引号和双引号等价
- \n, \t, \b, ', ", \, \v
- string is immutable: s[0] = 'n', error:'str' object does not support item assignment

```python
my_string = "Hello Byron"
print(my_string + my_string) # faster in Python3, enlarge the buffer, not copy to new.

'''multiple lines comments:
1. comments
2. comments
'''

print(my_string * 2)
print("="*20)
```

```
Hello ByronHello Byron
Hello ByronHello Byron
====================
```

```python
my_string = "Hello Byron"
print(len(my_string))
my_string[0] = 'a' # immutable
```

```
11

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-3-cb04cd8aa119> in <module>
      1 my_string = "Hello Byron"
      2 print(len(my_string))
----> 3 my_string[0] = 'a' # immutable
```

```
TypeError: 'str' object does not support item assignment # 不可更改
```

```python
# String Slice 留头不留尾
print(my_string[0])
print(my_string[0:3])
print(my_string[:-1])
print(my_string[-1])
print(my_string[0:6:2]) # 0到6 逢2提取 -> 0,2,4
print(my_string[::-1]) # 倒序
```

```
H
Hel
Hello Byro
n
Hlo
noryB olleH
```

```python
my_string = "Hello Byron "
print(my_string)
print(my_string.upper()) # 大写
print(my_string.lower()) # 小写
print(my_string.capitalize()) # 首字母大写
print(my_string.count('o')) # 数'o'的个数
print(my_string.strip()) # for '\n','\t', ' '; lstrip, rstrip 去除首尾空格
print(my_string.replace('n', 'ner')) # 替代
```

```
Hello Byron
HELLO BYRON
hello byron
Hello byron
2
Hello Byron
Hello Byroner
```

```python
print(my_string)
print(my_string.strip())
```

```
 Test Python
Test Python
```

```python
# More methods
#dir(my_string)
print(my_string.index('H'))
print(my_string.find('By')) # 返回位置
```

```
06
```

```python
# String format
"Hi, %s %2.1f" % ('Python', 2.712) # %2占位符 2格
```

```
'Hi, Python 2.7'
```

```python
print("%+10x" % 10) # 占位10格
print("%04d" % 5) # 补0直至4位
print("%6.3f" % 2.3) # 占6位，保留3位小数
```

```
        +a
0005
2.300
```

```
id = 123name = 'Byron'"HI, the id:{} of name:{}".format(id, name) # format 格式化
```

```
'HI, the id:123 of name:Byron'
```

```
L = ['This', 'is', 'a', 'good', 'person']
print("_".join(L))
```

```
This_is_a_good_person
```

List

- list = ['str', 2, ...]
- list[0]

```
# Create a list
mylist = ["my", "list", 1,2.3]
print(mylist)
print(mylist[0])
print(mylist[-1])
print(mylist[0:2])
print(mylist[2:])
print(mylist[:])
```

```
['my', 'list', 1, 2.3]
my
2.3
['my', 'list']
[1, 2.3]
['my', 'list', 1, 2.3]
```

```
## List of lists, NOTED: Row First! same as C
## Matlab: Column First!
house = [['living room', 25],
         ['bed room', 13],
         ['bathroom', 9.1]]
print(house[0])
print(house[1][1])
print(house[:])
```

```
['living room', 25]
13
[['living room', 25], ['bed room', 13], ['bathroom', 9.1]]
```

```
## List Operations
x = ["a", "b", "c", "d"]
print(x)
del(x[1])
print(x)
x = x + ['e']  ## what about? x = x + 'e' -> Error!
print(x)

x.append('e2')
print(x)
```

```
['a', 'b', 'c', 'd']
['a', 'c', 'd']
['a', 'c', 'd', 'e']
['a', 'c', 'd', 'e', 'e2']
```

```
## Copy list !!!Should be NOTED!
x = ["a", "b", "c", "d"]
y = x
z = x[:]
print(x, y, z)
x[0] = 'A'
print(x, y, z)
# x与y有同一套地址，共用一个储存空间，所以会一起变
# 但z是另外一套地址，所以不受影响
```

```
['a', 'b', 'c', 'd'] ['a', 'b', 'c', 'd'] ['a', 'b', 'c', 'd']
['A', 'b', 'c', 'd'] ['A', 'b', 'c', 'd'] ['a', 'b', 'c', 'd']
```

```
## function, method on list
x = ["a", "b", "c", 2]
print(len(x)) # 返回list长度
print(type(x[0]), type(x[3]))

y = sorted(x[0:3], reverse=True)## NOTICE: x is not changed!
print(y)

x.append('e')
print(x)
```

```
4
<class 'str'> <class 'int'>
['c', 'b', 'a']
['a', 'b', 'c', 2, 'e']
```

Tuples

- Tuple = (123, 'str', ...), T = (1,), not (1)
- Tuple[0]
- Values in Tuple are Immutalble!! 固定 数据不能动

```
## Define a Tuple ()
T = (123, 45, 'hello')
print(T[0])
T[0] = 234
print(T[0])
```

```
123

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-54-efd705126caa> in <module>
      2 T = (123, 45, 'hello')
      3 print(T[0])
----> 4 T[0] = 234
      5 print(T[0])
```

```
TypeError: 'tuple' object does not support item assignment # 不能更改
```

```
## Tuple Packing and Unpacking
T = 1, 'str', True
x, s, b = T
print(s)
```

```
str
```

```
## interchange the x,y
x = 3
y='a'
x, y = y, x # 一步交换print(x, y)x, y = 3, 'b'print(x, y)
```

```
a 3
3 b
```

```
## Tuple is immutable, however, could be mutable using a list
T = ('A', 'B', ['A', 'B'])
print(T[0])
print(T[2])
T[2][0] = 'X'
print(T[2])
```

```
A
['A', 'B']
['X', 'B']
```

Sets

- set = {'e1', 'e2', ..}
- set = set('strings')
- 会去掉重复量
- 无序

```
## element in Set is Unique, Unordered
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)
print('apple' in basket)
```

```
{'banana', 'pear', 'apple', 'orange'}
True
```

```python
## Set Operations: Union, Intersection, Difference, etc
a = set('abracadabra')
b = set('alacazam')
print(a, b)

print('Set Union: ', a|b) # 并集
print('Set Intersection: ', a&b) # 交集
print('Set Difference: ', a-b) # 差值
print('Set Symmetric Difference: ', a^b)  ## letters in a or b but not both #只有a有或者只有b有
print('Set ymmetric Difference: ', (a|b) - (a&b))
```

```
{'r', 'c', 'b', 'd', 'a'} {'c', 'l', 'm', 'z', 'a'}
Set Union:  {'r', 'c', 'l', 'm', 'z', 'b', 'd', 'a'}
Set Intersection:  {'a', 'c'}
Set Difference:  {'b', 'd', 'r'}
Set Symmetric Difference:  {'z', 'b', 'r', 'd', 'l', 'm'}
Set ymmetric Difference:  {'r', 'l', 'm', 'z', 'b', 'd'}
```

Dictionary

- dict = {key: value, ...}
- dict['key']

```python
## Define a dictionary
europe = {'spain':'madrid', 'france':'paris',
          'germany':'berlin','norway':'oslo'}
print(europe['france'])
print(europe.keys())
```

```
paris
dict_keys(['spain', 'france', 'germany', 'norway'])
```

```python
## KEY Should be Unique
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin','norway':'oslo'}
print(europe['spain'])
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin',
'norway':'oslo','spain':'****'}
print(europe['spain']) # 两个相同的keys

## Any immutable values can be the KEY
dict = {0:'value1', True:'value2', 12:14}
print(dict[0], dict[True], dict[12])
```

```
madrid
****
value1 value2 14
```

```python
## Operations on Dictionary
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo'}
#### add item in dict
europe['italy'] = 'rome'
print('italy' in europe)

#### update item in dict
europe['spain'] = 'Madrid'

#### remove item in dict
del(europe['spain'])

print(europe)
```

```
True
{'france': 'paris', 'germany': 'berlin', 'norway': 'oslo', 'italy': 'rome'}
```

```python
## Dictionary in Dictionary
europe = { 'spain': { 'capital':'madrid', 'population':46.77 },
           'france': { 'capital':'paris', 'population':66.03 },
           'germany': { 'capital':'berlin', 'population':80.62 },
           'norway': { 'capital':'oslo', 'population':5.084 } }
print(europe['france']['capital'])

data = {'capital':'rome', 'population':59.83}
europe['italy'] = data
print(europe)
```

```
paris
{'spain': {'capital': 'madrid', 'population': 46.77}, 'france': {'capital': 'paris', 'population': 66.03}, 'germany': {'capital': 'berlin', 'population': 80.62}, 'norway': {'capital': 'oslo', 'population': 5.084}, 'italy': {'capital': 'rome', 'population': 59.83}}
```

Comprehensions

```python
## List Comprehensions
squares = []
for x in range(10):
    squares.append(x**2)
print(squares)

#### others:
squares = list(map(lambda x: x**2, range(10)))
print(squares)
squares = [x**2 for x in range(10)]
print(squares)

## Complex expressions
from math import pi
mylist = [str(round(pi, i)) for i in range(1, 6)]
print(pi, '\nRound:', mylist)

## Nested List Comprehensions
matrix = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
print(matrix)
print('Transpose of Matrix:\n', [[row[i] for row in matrix] for i in range(4)])
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
3.141592653589793
Round: ['3.1', '3.14', '3.142', '3.1416', '3.14159']
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
Transpose of Matrix:
 [[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

```python
matrix = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
for i in range(4):
    for row in matrix:
        print(row)
        print(row[i])
```

```
[1, 2, 3, 4]
1
[5, 6, 7, 8]
5
[9, 10, 11, 12]
9
[1, 2, 3, 4]
2
[5, 6, 7, 8]
6
[9, 10, 11, 12]
10
[1, 2, 3, 4]
3
[5, 6, 7, 8]
7
[9, 10, 11, 12]
11
[1, 2, 3, 4]
4
[5, 6, 7, 8]
8
[9, 10, 11, 12]
12
```

```python
## Use conditions in comprehensions
UserName = ["peter", "Jonson", "Winter Ding"]
new_list = [name for name in UserName if len(name) > 5]
print(new_list)

new_list2 = [name if len(name) > 5 else "NULL" for name in UserName]
print(new_list2)
```

```
['Jonson', 'Winter Ding']
['NULL', 'Jonson', 'Winter Ding']
```

Generate matrix using List Comprehension

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

```
My_Matrix = [[(i+j)%4 + 1 for j in range(4)] for i in range(4)]
print(My_Matrix)

## OR:

a = [i for i in range(1,5)]*2
My_Matrix = [a[i-1:i+3] for i in range(1,5)]
print(My_Matrix)

list(map(lambda s: print(s), [a[i-1:i+3] for i in range(1,5)]))
```

```
[[1, 2, 3, 4], [2, 3, 4, 1], [3, 4, 1, 2], [4, 1, 2, 3]]
[[1, 2, 3, 4], [2, 3, 4, 1], [3, 4, 1, 2], [4, 1, 2, 3]]
[1, 2, 3, 4]
[2, 3, 4, 1]
[3, 4, 1, 2]
[4, 1, 2, 3]

[None, None, None, None]
```

```
## Set Comprehensions
a = {x for x in 'abcdefghacd' if x not in 'abc'}
print(a)
```

```
{'e', 'd', 'g', 'f', 'h'}
```

```
## Dictionary Comprehensions
dict = {x:x**2 for x in (2,3,4)}
print(dict)
```

```
{2: 4, 3: 9, 4: 16}
```

Generator Expression

- (), instead of [], in List Comprehension

```python
result = (num for num in range(10))
print(result)
print("Next value in Generator:", next(result))
for i in result:
    print(i)
```

```
<generator object <genexpr> at 0x7fec77588190>
Next value in Generator: 0
1
2
3
4
5
6
7
8
9
```

More

Loops in Dictionary

```python
dict = {'a':1, 'b':2, 'c':3}
for k, v in dict.items():
    print(k,v)
print('')

## Print index and corresponding values using enumerate() function
for i, v in enumerate(['a', 'b', 'c']):
    print(i, v)
print('')

## match list using zip() function
L1 = ['a', 'b', 'c']
L2 = [1, 2, 3]
for k, v in zip(L1, L2):
    print(k, v)
```

```
a 1
b 2
c 3

0 a
1 b
2 c

a 1
b 2
c 3
```

Matplotlib

- Simple Line, Scatter figures with properties in Label, Size, Color, Ticks

```python
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [1.1, 2.1, 3.1, 4.1, 5.1]
plt.plot(x,y)
plt.show()
```

```python
import numpy as np
col = ['red', 'green', 'blue', 'yellow', 'black']
plt.scatter(x, y, s = np.array(y)*200, c = col, alpha = 0.8)
## s: size array;  c: color list;  alpha: transparency

plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Title')
plt.yticks([1, 2, 3, 4, 5], ['1-K', '2-K', '3-K', '4-K', '5-K'])
plt.show()
```

List VS Tuple

```
## List: list is dynamical
## Tuple: tuple is static
l = [1,2,3]
print(l.__sizeof__()) # 存储空间
t = (1,2,3)
print(t.__sizeof__())
```

```
6448
```

```
## over-allocating: reserve more memory for efficiency
## preserve 4*8(int) = 32 bytes
## Computational complexity: O(1) for Adding new or detete
l = []
l.__sizeof__()
```

```
40
```

```
l.append(1)
print(l)
l.__sizeof__()
```

```
[1]
72
```

```
l.append(2) # 放1的时候，预先扩充了4个位置
print(l)
l.__sizeof__()
```

```
[1, 2]
72
```

```
l.append(3)
print(l)
l.__sizeof__()
```

```
[1, 2, 3]
72
```

```python
l.append(4)
print(l)
l.__sizeof__()
```

```
[1, 2, 3, 4]
72
```

```python
l.append(5)
print(l)
l.__sizeof__()
```

```
[1, 2, 3, 4, 5]
104
```

```python
type(l[1])
```

```
int
```

Efficiency: Tuple is faster than List

```python
%timeit x=[1,2,3,4,5,6]
```

```
57.3 ns ± 0.59 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

```python
%timeit x=(1,2,3,4,5,6)
```

```
13.1 ns ± 0.331 ns per loop (mean ± std. dev. of 7 runs, 100000000 loops each)
```

Dict and Set

- They are Hash Table! (https://en.wikipedia.org/wiki/Hash_table#/media/File:Hash_table_5_0_1_1_1_1_1_LL.svg)
    - Dict: a table to record: Hash, key and value [hash is calculated from key]
    - Set: a table to record: Hash, value [hash is calculated from value]
- Faster than List for adding or searching in O(1), compared to List in O(n^2) 直接计算hash值，找到所在位置

NOTICE:

- Dict.pop(): delete the last element
- Set.pop(): not sure as the element is unordered in set.

```python
## 打印九九乘法表
for i in range(1,10):
    for j in range(1,i+1):
        print(str(j) + '*' + str(i) + '=' +str(i*j)+'\t', end='')
    print()
```

```
1*1=1
1*2=2 2*2=4
1*3=3 2*3=6 3*3=9
1*4=4 2*4=8 3*4=12  4*4=16
1*5=5 2*5=10  3*5=15  4*5=20  5*5=25
1*6=6 2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7 2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8 2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9 2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

Shallow copy

```python
l1 = [[1,2],(30,40)]
l2 = list(l1)
l1, l2
```

```
([[1, 2], (30, 40)], [[1, 2], (30, 40)])
```

```python
l1 is l2
```

```
False
```

```
l1 == l2
```

```
True
```

```
l1.append(100)
l1[0].append(3)
l1[1] += (50,60)
l1,l2 # list变了, tuple没变
```

```
([[1, 2, 3], (30, 40, 50, 60), 100], [[1, 2, 3], (30, 40)])
```

Control Flow Print

Print

```
## Print String

s = "hello"
print("String=%s" % s)
print("String=%10s" % s)  ## ADD SPACE ON THE LEFT SIDE 控制占位符是10, 右对齐
print("String=%-10sEND" % s) ## 控制占位符是10, 左对齐
```

```
String=hello
String=     hello
String=hello     END
```

```
## Print Int
x = 5
print('Num =%d' % x) ## digits
print('Num =%5d' % x)
print('Num =%-5dEND' % x)
print('Num =%05d' % x) ## 前面补0
## Complex setting
print('Num =%*.*d' % (5,4,x)) ## *占位符, *补充够多少数字
print('Num =%*.*d' % (5,4,789))
```

```
Num =5
Num =     5
Num =5     END
Num =00005
Num = 0005
Num = 0789
```

```python
## Print Float
import math

print('PI = %f' % math.pi)
print('PI = %10.3f' % math.pi)
print('PI = %-10.3fEND' % math.pi)
```

```
PI = 3.141593
PI =      3.142
PI = 3.142     END
```

```python
## Print With Format
print('%r %r %r' % (1, 2, 3)) ## %r: raw strings
print('%r %r %r' % ('one', 'two', 'three'))

## Print With Position Index
A = 'XP'
B = 'Hello!'
print("{1}, my name is {0}".format(A, B)) # 大括号
## Print With Position Index
print("{HI}, my name is {NAME}".format(NAME=A, HI=B))
## Print With List
person = ['XP', 'Hello!']
print('{PEOPLE[1]}, my name is {PEOPLE[0]}'.format(PEOPLE = person))
```

```
1 2 3
'one' 'two' 'three'
Hello!, my name is XP
Hello!, my name is XP
Hello!, my name is XP
```

```
## Define a Template
FormatSTR = "{1} and {0} are string types"
FormatSTR.format(A, B)
```

```
'Hello! and XP are string types'
```

```
## USE f-string 格式化字符串常量
## \n 换行
C = 1.234
D = 1234
S = f"{B.replace('ello!', 'i')}, my name is '{A}'! \nRound float: {C:.2f} and Exponential Format: {D:e}" # 科学计数法
print(S)
```

```
Hi, my name is 'XP'!
Round float: 1.23 and Exponential Format: 1.234000e+03
```

```
## Format OUTPUT
a = 123.456789
print('Float Number:{0:4.2f}'.format(4.1234)) ## 0 指代第一个数
s = 'abc'
print('First  method: Number: {:2.1f} and string: {}'.format(24.12, 1231.123))
print('Second method: Number: %2.1f and string: %s' % (24.12, 1231.123))

## align : ^, <, > For Center, Left, Right alignment
print("{0:0.3f} VS ${1:<10s}$".format(a, s)) # 左对齐
print("{0:2.2f} VS ${1:>10s}$".format(a, s)) # 右对齐
print("{0:5.5f} VS ${1:^10s}$".format(a, s))  ## 居中 Question: Where is '8'? 四舍五入
```

```
Float Number:4.12
First  method: Number: 24.1 and string: 1231.123
Second method: Number: 24.1 and string: 1231.123
123.457 VS $abc       $
123.46 VS $       abc$
123.45679 VS $   abc    $
```

IF-ELIF-ELSE

```python
x = 7
if x%2 == 0:
    print('Even Number')
elif x%3 == 0:
    print('ODD Number but divided by 3')
else:
    print('Odd number')
```

```
Odd number
```

LOOPS

```python
## While loops
x = 1
while x<5:
    print(x)
    x = x + 1  ## 等价于: x += 1
```

```
1
2
3
4
```

```python
## Check x += 1 && x =+ 1
x = 1
x =+ 1
print(x)
x += 1   #x = x + 1
print(x)
```

```
1
2
```

```python
## simple iterations using: range()
for i in range(10): # [0,9] 不包括10
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```python
## Loop Over String
for c in "python":
    print(c.capitalize())
```

```
P
Y
T
H
O
N
```

```python
## Loop Over List
areas = [1, 2, 3, 4]
for item in areas:
    print(item)

## With enumerate
for index, value in enumerate(areas):
    print(index+1, value)
```

```
1
2
3
4
1 1
2 2
3 3
4 4
```

```python
## Loop Over Dictionary
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin',
          'norway':'oslo', 'italy':'rome', 'poland':'warsaw', 'austria':'vienna' }

# Iterate over europe
for key, value in europe.items():
    print("The capital of " + key + " is " + value)

print(europe.keys())

for key in europe.keys():
    print("The capital of " + key + " is " + europe[key])
```

```
The capital of spain is madrid
The capital of france is paris
The capital of germany is berlin
The capital of norway is oslo
The capital of italy is rome
The capital of poland is warsaw
The capital of austria is vienna
dict_keys(['spain', 'france', 'germany', 'norway', 'italy', 'poland', 'austria'])
The capital of spain is madrid
The capital of france is paris
The capital of germany is berlin
The capital of norway is oslo
The capital of italy is rome
The capital of poland is warsaw
The capital of austria is vienna
```

```python
## Loop Over Numpy Array
import numpy as np

## Loop over 1-d array
np_array = np.array([1, 2, 3, 4])
for x in np_array:
    print(x)

dl = [[1,2,3], [2,3,4], [3,4,5]]
np_array2d = np.array(dl)
print(np_array2d)

## Loop over 2-d array, Go through each element in 2d array
for item in np.nditer(np_array2d):
    print(item)

for item in np_array2d:
    print(item)
```

```
1
2
3
4
[[1 2 3]
 [2 3 4]
 [3 4 5]]
1
2
3
2
3
4
3
4
5
[1 2 3]
[2 3 4]
[3 4 5]
```

Functions

```
## Simple example
def fun_name(var1='default value '):
    var = var1*3
    print(var)

fun_name('test ')
fun_name()
```

```
test test test
default value default value default value
```

```
## Nested Functions: NOTE: SCOPE
def three_shouts(word1, word2, word3):
    """Returns a tuple of strings
    concatenated with '!!!'."""

    # Define inner
    def inner(word): # 内置函数
        """Returns a string concatenated with '!!!'."""
```

```python
        return word + '!!!'

    # Return a tuple of strings
    return (inner(word1), inner(word2), inner(word3))

# Call three_shouts() and print
print(three_shouts('a', 'b', 'c'))
```

```
('a!!!', 'b!!!', 'c!!!')
```

```python
## Nested functions
def echo(n):
    """Return the inner_echo function."""

    # Define inner_echo
    def inner_echo(word1):
        """Concatenate n copies of word1."""
        echo_word = word1 * n
        return echo_word

    # Return inner_echo
    return inner_echo

# Call echo: twice
twice = echo(2)

# Call echo: thrice
thrice = echo(3)

# Call twice() and thrice() then print
print(twice('hello'), thrice('hello'))
```

```
hellohello hellohellohello
```

```python
## Keyword:'nonlocal'  in inner nested function
## Questions: Why not global?
## global 面对全局变量；而nonlocal面对外层变量，即echo_shout()内

# Define echo_shout()
def echo_shout(word):
    """Change the value of a nonlocal variable"""

    # Concatenate word with itself: echo_word
    echo_word = word * 2

    # Print echo_word
    print(echo_word)

    # Define inner function shout()
    def shout():
        """Alter a variable in the enclosing scope"""
        # Use echo_word in nonlocal scope
        nonlocal echo_word

        # Change echo_word to echo_word concatenated with '!!!'
        echo_word = echo_word + '!!!'

    # Call function shout()
    shout()

    # Print echo_word
    print(echo_word)

# Call function echo_shout() with argument 'hello'
echo_shout('hello')
```

```
hellohello
hellohello!!!
```

```python
## Functions with variable-length arguments (*args)

def CatStr(var1, *args): ## CatStr(*args)
    allstr = ""
    for s in args:
        allstr += s
    return allstr
print(CatStr("Hi, ", "How ", "are ", "you "))
```

```
How are you
```

```python
## Functions with variable-length Keyword arguments (**kwargs) 任意个数任意变量

# Define report_status
def report_status(**kwargs):
    """Print out the status of a movie character."""

    print("\nBEGIN: REPORT\n")

    # Iterate over the key-value pairs of kwargs
    for keys, values in kwargs.items():
        # Print out the keys and values, separated by a colon ':'
        print(keys + ": " + values)

    print("\nEND REPORT")

# call to report_status()
report_status(name="luke", affiliation="jedi", status="missing")
```

```
BEGIN: REPORT

name: luke
affiliation: jedi
status: missing

END REPORT
```

Lambda Funtion

- anonymous function

```python
def echo_word(word1, echo):
    """Concatenate echo copies of word1."""
    words = word1 * echo
    return words
print(echo_word('hey ', 5))

## Using lambda function
echo_word = (lambda word1, echo: word1 * echo)
print(echo_word('hey ', 5))
```

```
hey hey hey hey hey
hey hey hey hey hey
```

```python
## lambda function with no arguments
fun = (lambda : print('Hi'))
fun()
```

```
Hi
```

Lambda + Map

map() 将可迭代的变量输送到前面 需要lambda返回值

```
nums = [2, 4, 6, 8, 10]
result = map(lambda a: a ** 2, nums) #generator
print(result)
for i in result:
    print(i)
print(list(result))
```

```
<map object at 0x7fc97239edc0>
4
16
36
64
100
[]
```

Lambda + Filter

Filter() 过滤器 需要lambda返回True/False

```
# Create a list of strings: fellowship
fellowship = ['frodo', 'samwise', 'merry', 'pippin', 'aragorn', 'boromir', 'legolas', 'gimli', 'gandalf']

# Use filter() to apply a lambda function over fellowship: result
result = filter(lambda member: len(member) > 6, fellowship)

# Convert result to a list: result_list
result_list = list(result)

# Print result_list
print(result_list)
```

```
['samwise', 'aragorn', 'boromir', 'legolas', 'gandalf']
```

Lambda + Reduce

reduce() 累加效果，需要两个变量。取出x，y，算出来的值赋值给x，然后再往后取x，y

```python
## Reduce function for accumulative calculation
## reduce() 函数会对参数序列中元素进行累积。
## 函数将一个数据集合（链表，元组等）中的所有数据进行下列操作：
## 用传给 reduce 中的函数 function（有两个参数）先对集合中的第 1、2 个元素进行操作，得到的结果再与第三个数据用 function 函数运算，最后得到一个结果。

from functools import reduce
reduce(lambda x, y: x+y, [1,2,3,4,5])
```

```
15
```

```python
# Create a list of strings: stark
stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon']

# Use reduce() to apply a lambda function over stark: result
result = reduce(lambda item1, item2: item1 +" "+ item2, stark)

# Print the result
print(result)
```

```
robb sansa arya brandon rickon
```

Error-Handing with Try-Except

```python
try:
    fh = open("testfile2", "r")
except IOError:
    print("Error: File is NOT FOUND!!!")
else:
    print("Successfully Reading!")
    fh.close()
```

```
Error: File is NOT FOUND!!!
```

Class

```python
##### Define  a class [inherit object(default)]
class Student(object):
    def __init__(self, name, score):
        self.name = name
        self.__score = score  ### Hidden variable, can't visit by outside method
        self.__score__ = score

    def print_score(self):
        print('%s: %s' % (self.name, self.__score))

    def get_grade(self):
        if self.__score >= 90:
            return 'A'
        elif self.score > 80:
            return 'B'
        else:
            return 'C'

A = Student('Lu', 90)
A.print_score()
```

```
Lu: 90
```

```
A.get_grade
```

```
<bound method Student.get_grade of <__main__.Student object at 0x7fa0f24164c0>>
```

```
A.get_grade()
```

```
'A'
```

```
A.name = 'Wang'
A.__score #不可从外部访问属性
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-50-a96eeaba5d64> in <module>
----> 1 A.__score

AttributeError: 'Student' object has no attribute '__score'
```

```
A.__score__
```

```
90
```

```
A.name
```

```
'Wang'
```

end